usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

BCW: Buffer-Controlled Writes to HDDs for SSD-HDD Hybrid Storage Server

Shucheng Wang, Ziyi Lu, and Qiang Cao, *Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System;* Hong Jiang, *Department of Computer Science and Engineering, University of Texas at Arlington;* Jie Yao, *School of Computer Science and Technology, Huazhong University of Science and Technology;* Yuanyuan Dong and Puyuan Yang, *Alibaba Group*

https://www.usenix.org/conference/fast20/presentation/wang-shucheng

This paper is included in the Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST '20)

February 25–27, 2020 • Santa Clara, CA, USA

978-1-939133-12-0

Open access to the Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST '20) is sponsored by

NetApp®

BCW: Buffer-Controlled Writes to HDDs for SSD-HDD Hybrid Storage Server

Shucheng Wang[1], Ziyi Lu[1], Qiang Cao[1]*, Hong Jiang[3],
Jie Yao[2], Yuanyuan Dong[4] and Puyuan Yang[4]
[1]Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System,
[2]School of Computer Science and Technology, Huazhong University of Science and Technology,
[3]Department of Computer Science and Engineering, University of Texas at Arlington,
[4]Alibaba Group

Abstract

Hybrid Storage servers combining high-speed SSDs and high-capacity HDDs are designed for high cost-effectiveness and provide μs-level responsiveness for applications. Observations from the production hybrid cloud storage system Pangu suggest that HDDs are often severely underutilized while SSDs are overused, especially for writes that dominate the hybrid storage. This lopsided utilization between HDDs and SSDs leads to not only fast wear-out in the latter but also very high tail latency due to frequent garbage collections induced by intensive writes to the latter. On the other hand, our extensive experimental study reveals that a series of sequential and continuous writes to HDDs exhibit a periodic, staircase shaped pattern of write latency, i.e., low (e.g., 35μs), middle (e.g., 55μs), and high latency (e.g., 12ms), resulting from buffered writes in HDD's controller. This suggests that HDDs can potentially provide μs-level write IO delay (for appropriately scheduled writes), which is close to SSDs' write performance. These observations inspire us to effectively exploit this performance potential of HDDs to absorb as many writes as possible to avoid SSD overuse without performance degradation.

To achieve this goal, we first characterize performance behaviors of hybrid storage in general and its HDDs in particular. Based on the findings on sequential and continuous writes, we propose a prediction model to accurately determine next write latency state (i.e., fast, middle and slow). With this model, a Buffer-Controlled Write approach, BCW, is proposed to proactively and effectively control buffered writes so that low- and mid-latency periods in HDDs are scheduled with application write data and high-latency periods are filled with padded data. Based on BCW, we design a mixed IO scheduler (MIOS) to adaptively steer incoming data to SSDs and HDDs according to write patterns, runtime queue lengths, and disk status. We perform extensive evaluations under production workloads and benchmarks. The results show that MIOS removes up to 93% amount of data written to SSDs, reduces average and 99th-percentile latencies of the hybrid server by 65% and 85% respectively.

*Corresponding author. Email: caoqiang@hust.edu.cn

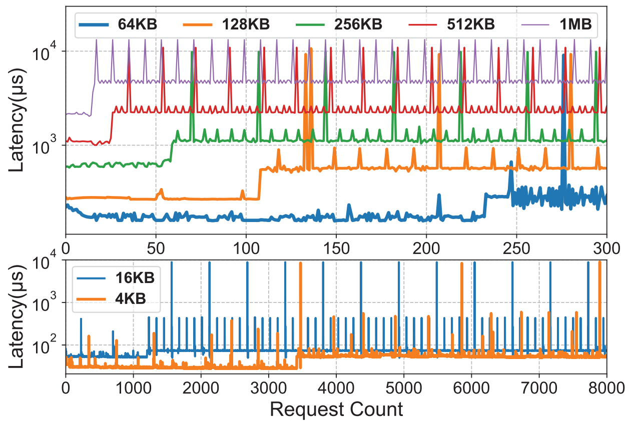


Figure 1: Sequential writing in a 10TB Western Digital HDD.

1 Introduction

Storage clouds have prevalently deployed hybrid storage servers integrating solid-state drives (SSDs) and hard-drive disks (HDDs) in their underlying uniform storage infrastructure, such as Alibaba Pangu [9], Amazon [38], Facebook [35], Google [20], Microsoft Azure [8]. Such hybrid storage servers employ an SSD-HDD tiered architecture to reap the benefits of both SSDs and HDDs for their superior IO performance and large capacity respectively, thus achieving high cost-effectiveness. Incoming writes are quickly persisted in the SSD tier and acknowledged, and then flushed to the HDD tier at a later time.

Our observations from real-world production workloads of hybrid storage servers in Pangu indicate that, SSDs are generally over-used while HDDs are less than 10% utilized on average, missing the opportunity to exploit HDDs' performance and capacity potentials. However, writes are known to be unfriendly to SSDs for two reasons. First, SSDs have limited P/E (Program/Erase) cycles [6, 36] that are directly related to the amount of writes. Second, SSDs suffer from unpredictable, severe performance degradations resulting from garbage collections [26, 51]. To guarantee stable write performance of storage servers in write-heavy workloads, cloud providers have to deploy more and/or larger SSDs, significantly increasing their total investment capital.

Our extensive experimental study on HDD write behaviors, conducted on various HDD products and with results shown in Figure 1, suggests that a series of continuous and sequential small HDD writes (e.g., 4KB) exhibit low latency (e.g., 35μs) for about 60 ms, and then a sharply elevated high latency (e.g., 12ms), which is followed by medium latency (e.g., 55μs) for about 40ms. The three states of write behaviors, or *write states* in short, are referred to in this paper as *fast, mid*, and *slow* writes, respectively. The former two types of writes can provide μs-level responsiveness, because incoming writes are considered complete and acknowledged (to the host) once their data have been written into the built-in buffer in the controller. However, when the write buffer is full, host writes have to be blocked until the buffered data are flushed into the disk, causing slow writes. This finding inspires us to fully exploit performance potentials offered by buffered writes of HDD, improving the performance while mitigating write-penalty on SSDs. Our goal is to enable hybrid storage servers to achieve higher performance and reliability without introducing extra hardware.

However, the key challenge for adopting buffered writes in HDDs to take advantage of the fast and mid writes is the difficulty in predicting precisely when these write states would occur. The internal buffer and other components of HDDs are completely hidden from the host. Host can only identify the current write state according to its own delay, but not future write states. To address this issue, we build a prediction model for sequential and continuous write patterns that predicts the next HDD write state. The insights is that, the write states of continuous and sequential HDD write requests is periodical. The prediction of next write state can be achieved with the information of buffered-write period and current write state. Then, we propose a Buffer-Controlled Write (BCW) approach. BCW can proactively and effectively control the buffer write behavior according to the predictor and runtime IO monitoring. Besides, BCW also actively “skip” slow writes by filling padded data during HDD slow writes.

We further propose a mixed IO scheduler (MIOS) for SSD-HDD hybrid storage by leveraging the BCW approach. MIOS adaptively redirects incoming writes to SSDs or HDDs depending on write states, runtime queue length, and disk status. Under high IO intensity, MIOS can be triggered to reduce IO pressure, the amount of data written and write penalty on SSDs while improving both average and tail latency.

The main contributions of this paper are as follows.

- Through extensive experimental studies on HDD write behaviors, we discover that there exist a periodic staircase-shaped write latency pattern consisting of μs-level write latency (low and mid write states) followed by ms-level write latency (slow write state) upon continuous and sequential writes, because of the buffered write feature in HDDs. To facilitate the full exploitation of this write latency pattern, we build a predictor to pre-determine what the next write state is.
- We propose a buffer-controlled write (BCW) approach, which proactively activates continuous and sequential write patterns, as well as effectively controls the IO behavior, according to the predictor and runtime IO monitoring. BCW also employs data padding to actively avoid, or skips, slow writes for the host.
- We design an SSD-HDD mixed IO scheduler (MIOS) to improve the overall performance of SSD-HDD hybrid storage servers, while substantially reducing write traffic to SSDs.
- We prototype and evaluate MIOS under a variety of production workloads. The results demonstrate that MIOS reduces average and tail latency significantly with dramatic decrease in the amount of data written to SSDs.

The rest of the paper is organized as follows. Section 2 provides the necessary background for the proposed BCW approach. Section 3 analyzes the behaviors of HDD buffered writes. Section 4 describes design and implementation of BCW and MIOS. We evaluate the effectiveness of MIOS in Section 5. Finally, Section 6 describes related works and Section 7 concludes the paper.

2 Background and Motivation

2.1 Primary Storage

Nowadays, primary storage involves popular solid-state driver (SSD), and traditional hard disk drive (HDD). SSDs have become a mainstream storage media due to its superior performance to and lower power consumption than HDDs [1, 49]. However, the limited write endurance has become a critical design issue in SSD-based storage systems [34]. Furthermore, SSDs suffer from performance-degrading garbage collections (GCs), which recycle the invalid pages by moving valid parts to new blocks and then erasing old blocks [27, 37]. GCs with ms-level delays can block incoming user requests, thus leading to long tail latency [17]. On the other hand, both large IO blocks and high IO intensity can lead to sudden increase in SSD queue, resulting in high tail latency [26]. Therefore, recent studies [50] indicate that SSDs do not always exhibit their ideal performance in practical.

HDDs have large capacity at low cost without the wear-out problem. However, HDDs have relatively low performance compared to SSDs. A random HDD IO has 2$\sim$3 orders of magnitude higher latency than an SSD IO. This is primarily because of the ms-level mechanical seeking of disk head.

2.2 SSD-HDD Hybrid Storage

To accommodate exponentially increasing storage requirement while achieving overall cost-effectiveness, SSD-HDD hybrid storage has emerged to be an inevitable choice for cloud providers [40, 47]. Most providers, such as Google [20], Amazon [38], Facebook [35], and Microsoft's online services [8], expect larger storage capacity and better performance but at lower cost. To meet this demand, they increasingly embrace storage heterogeneity, by deploying variable types and numbers of SSDs, which offer lower IO latency [14], as the primary tier and HDDs, which provide larger capacity at low cost as the secondary tier. The fast SSD tier generally plays the role of a write buffer to quickly persist incoming write data, which are eventually flushed to the slower but larger HDD tier. As a result, the SSD tier absorbs most of the write traffic from foreground applications.

2.3 Write Behavior of Hybrid Storage

Write-intensive workloads widely exist in many production environments, such as enterprise applications, supercomputing, and clouds. Enterprise servers are expected to rapidly persist production data in time, such as business databases. Burst buffer [3, 28] in supercomputing systems also deploy high-performance SSDs to temporarily store instantaneous highly-intensive write data.

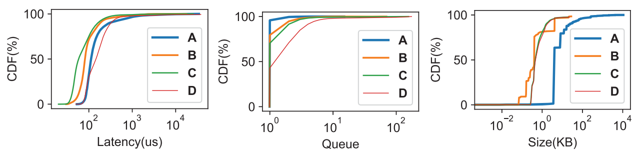
More commonly, many backend storage servers in cloud must accommodate write-dominated workloads, as observed in Alibaba Pangu [9]. Pangu is a distributed large-scale storage platform and provides cost-effective and unified storage services for Alibaba Cloud [22, 30] and Ant Financial [9]. As such, Pangu needs to minimize the total cost of ownership while meeting strict QoS requirements like tail latency [5, 15].

As an observation made through our analysis of production trace data from Pangu in Table 1, some storage nodes (servers) in Pangu rarely serve reads from the frontend and instead must handle amounts of highly-intensive writes. For Alibaba Cloud, because the upper-level latency-critical online services generally build their own application-aware caches to ensure service responsiveness and reserve local fast-storage to cache hot data for user reads, the backend storage nodes are thus required to persist new and updated data from frontend nodes as soon as possible. To better understand this write-dominated workload behavior, we analyze four typical workloads on Pangu storage nodes A (Cloud Computing), B (Cloud Storage), C and D (Structured Storage). We count the workloads from one SSD and one HDD in each node because the workload behavior of all storage devices is basically the same in one node. Observations are drawn as follows. A comprehensive workload analysis of Pangu can be found in the previous study [31].

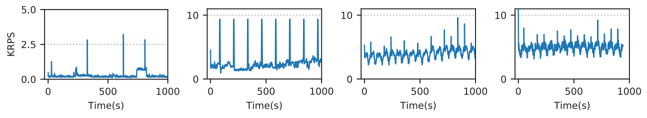
- Most requests are writes. As shown in Table 1, more than 77% and up to 95% of requests are writes in these nodes, and the amount of data written is 1-2 orders of magnitude larger than that of data read from them. Actually, nearly 3 TB data are written to every SSD each day, which is close to DWPD (Drive Writes Per Day) that strictly limits the amount of SSD data written daily for reliability.
- The IO intensity distribution has bursty patterns. As shown in Figure 2(d) through Figure 2(g), SSDs experience bursty intensive write workloads (e.g., 11K request per second in workload D).
- The amount of data written to SSDs and HDDs differ dramatically. For instance, the average SSD IO utilization is up to 28.5% in load B while it is less than 10% in HDD. Even so, most of the HDD utilization is used in dumping SSD data, rarely servicing user requests.
- There exists long tail latency. As shown in Figure 2(a), SSDs with high IOPS suffer from heavy-tail IO latency (e.g., the 99^{th} percentile latency is 10ms) due to queue blocking in Figure 2(b). This is caused in part by (1) large writes (e.g., 1MB), and (2) frequent garbage collections induced by high write intensity.
- Small size IOs account for a large proportion of all IOs. As shown in Figure 2(c), more than 75% of write requests are of 10KB or smaller, and the average request size is nearly 4KB in C and D.

Table 1: The workload characteristics of Pangu traces recorded from one SSD and one HDD in four different nodes, A B C and D, that support online services.

Node Type	Duration (min)	Writes (GB)	Reads (GB)	Avg. Req. Size(KB)	Peak KRPS	Avg. KRPS	Avg. HDD IO Uti.(%)	Avg. SSD IO Uti.(%)
A	45	18.5	1.4	56.0	3.4	0.23	7.6	11.9
B	30	74.4	2	17.7	9.3	2.5	9.8	28.5
C	30	10.7	2.1	4.2	9.6	2.7	4.1	24.6
D	26	10.1	1.7	4.1	11.1	3	4.8	25



(a) Latency (b) Queue Length (c) IO size

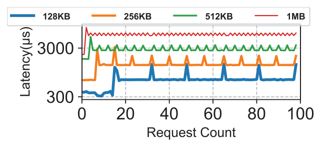


(d) Node A (e) Node B (f) Node C (g) Node D

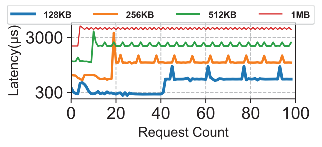
Figure 2: Behaviors of production workloads on four representative hybrid storage nodes in Pangu in terms of latency, queue length, request size and IO intensity.

2.4 Challenge

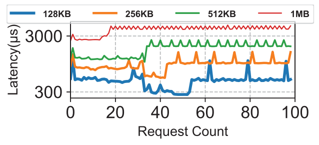
To relieve the SSD pressure from write-dominate workloads, a simple solution is to increase the number of SSDs in the hybrid nodes. However, this is a costly solution as it increases the total cost of ownership. An alternative is to exploit the severely underutilized HDD IO capacity in hybrid storage nodes when SSDs are overused. The state-of-art solution SSD-Write-Redirect (SWR) [31] redirects large SSD



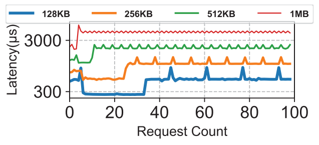
(a) 4TB West Digital HDD



(b) 4TB Seagate HDD



(c) 8TB West Digital HDD



(d) 8TB Seagate HDD

Figure 3: Sequential writing on four types of HDDs.

writes to idle HDDs. This approach can alleviate the SSD queue blocking issue to some extent. However, the IO delays experienced by requests redirected to HDDs are shown to be 3-12 times higher than those experienced on SSDs. This is clearly undesirable, if not unacceptable, for most small writes that demand μs-level latency. The key challenge is how to reduce HDD IO delay to the μs-level that is close to SSDs, a seemingly impossible task at a first glance. Fortunately, as we look closer into the write behaviors of HDDs, this is indeed a possible task, which we elaborate next.

3 HDD Write Behaviors

To have a comprehensive understanding HDD write behaviors, so as to assess the possibility of achieving μs-level write latency on HDDs, we perform continuous and sequential writes, which is the most friendly write pattern for HDDs.

3.1 Buffered Writes

We conduct a "*Profiling*" process to observe detailed HDD behaviors, a series of continuous and sequential writes with the same IO size are written to an HDD. We select five representative HDD products: West Digital 10TB (WD100EFAX [13]), 8TB (WD8004FRYZ [12]), 4TB (WD40EZRZ [13]), Seagate 8TB (ST8000DM0004 [44]), 4TB (ST4000DM004 [45]). The 4TB Seagate HDD is SMR (Shingled Magnetic Recording) and the other four HDDs are PMR (Perpendicular Magnetic Recording). We draw three interestng observations from the profiling results shown in Figure 1 and Figure 3.

- For each tested HDD, the series of continuous sequential write requests experience a similar sequence of three-level write latency, i.e., low, mid, and high latencies, forming a staircase-shaped time series. For example, in the 10TB HDD, the three levels of write latency of 16KB writes are about 66μs, 135μs, and 12ms respectively.
- The observed HDD write behavior is periodic. At the beginning (right after the buffer becomes empty) low-latency writes last for a period (e.g., 60ms in 10TB), which is followed by a spike (high-latency writes) and then mid-latency writes. If the write pattern is continuous, high-latency writes and mid-latency writes will appear alternately.
- The number of low-latency continuous writes in a sequence relies on their I/O size. Smaller write size leads to a larger number of writes. For example, the number of 16KB and 64KB writes is about 1200 and 240 on the 10TB HDD, respectively.

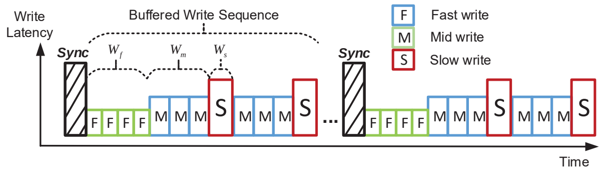


Figure 4: The HDD Buffered-Write Model with two complete Buffered Write Sequences.

The reasons behind these observed HDD write behaviors are as follows. Modern HDDs deploy a built-in DRAM (e.g., 256MB for the 10TB and 8TB HDDs, and 64MB for the two 4TB HDDs). However, only a part of the DRAM (e.g., 16MB for 10TB WD and 8TB Seagate HDD, 4MB for 8TB WD HDD and 4TB Seagate HDD, 2MB for 4TB WD HDD) can be used to buffer incoming write IOs based on external observation. The remaining capacity of the HDD built-in DRAM can be used as read-ahead cache, ECC buffer [10], sector remapping buffer, or prefetching buffer [18,43]. However, the exact mechanism by which this built-in DRAM in HDD is used, which varies with the HDD model, is generally proprietary to the HDD manufactures only. Fortunately, the actual size of write buffer can be measured externally by profiling.

Upon successful buffering of a write, HDD immediately informs the host of request completion. When the buffered data exceed a threshold, the HDD must force a flushing of the buffered data into their locations in the disk media. During this period, incoming writes must be blocked until the buffer is freed up again. It is worth noting that, after an idle period, the HDD buffer may become empty implicitly as a course of flushing data to the disk. However, to explicitly empty the buffer, we can actively invoke $sync()$ to force flushing.

3.2 An HDD Buffered-Write Model

To formally characterize the HDD write behavior, we build an HDD buffered-write model. Figure 4 illustrates the schematic diagram of the HDD buffered-write model in the time dimension (runtime). The x axis represents the time sequences of transitions among the three write levels, with each sequence being started by "Sync" event.

A Buffered-Write Sequence consists of three aforementioned types of HDD buffered writes, i.e., Fast (low-latency), Mid (mid-latency) and Slow (high-latency) writes, which denotes as F, M, and S, respectively. In the model, F, M, and S can be thought of as the states a write request can be in (i.e., experiencing the fast, mid or slow write process). As described in Table 2, these IO delays are denoted as L_f, L_m and L_s, respectively. The F state means that an incoming write

Table 2: The list of descriptions about all the parameters in the HDD Buffered-Write Model.

Parameters	Description
$L_{f/m/s}$	The IO delays of write requests in the F/M/S write states
$W_{f/m/s}$	The cumulative amount of written data for the Fast/Mid/Slow Stages
$T_{f/m/s}$	The time duration of the Fast/Mid/Slow Stages
s_i	The IO size of write request i

request w_i with the size of s_i can be buffered completely in the built-in DRAM buffer of HDD. The M state means that the write buffer is close to be full. The S state means that the write buffer is full and any incoming write request is blocked.

A Buffered Write Sequence lasts a Fast stage, followed by one or more Slow-and-Mid stage-pairs. The sequence begins when there is sufficient buffer available for Fast stage (e.g., close to empty). It ends when current series of continuous writes ends. The Fast, Mid, and Slow Stage last for T_f, T_m, and T_s respectively, which are determined by the cumulative amount of written data W_f, W_m, and W_s in the respective states. Actually, $W_f = T_f * s_i / L_f$ and it is applied to W_m.

We can profile the HDDs to identify such key parameters. For example, in the 10TB HDD with 64KB write requests shown in Figure 1, the value of L_f is 180μs, L_m is 280μs and L_s is 12ms. The value of T_f is 60 ms, T_m is 37ms and T_s is 12ms. W_f is 16MB and W_m is 8MB. W_s depends on the IO size s_i. According to the HDD buffered-write model, the Fast and Mid writes of HDD have 100-μs-level latency, which can approach the write latency of SSDs. This motivates us to design a controllable buffer write strategy for HDDs to reduce writes on SSDs in hybrid storage systems without sacrificing the overall performance.

Note that the buffering and flushing mechanisms are completely hidden from the host and heavily depend on the specific HDD models. Fortunately, we can measure the buffered write feature of an HDD externally and experimentally based on the aforementioned experiments.

4 Design

To fully exploit HDD buffered writes, two critical challenges must be addressed. The first is how to determine which write state that a write request will be Fast (F), Mid (M), or Slow (S), in order to properly schedule the write request. The second is how to steer an incoming write request to HDD without performance degradation.

For the first problem, we build a *Write-state Predictor* to pre-determine the next write state based on the current write state and buffer state. The ability to determine the subsequent write state of HDD is critical to scheduling incoming write requests. Based on that, we propose *Buffer-Controlled Writes*, shortly for **BCW**, a writing approach to proactively activate continuous and sequential write patterns that the predictor relied on, as well as effectively controls the IO behavior according to the predictor and runtime IO monitoring. To avoid performance degradation caused by S writes, we propose a proactive padding-write approach to “skip” the S state by

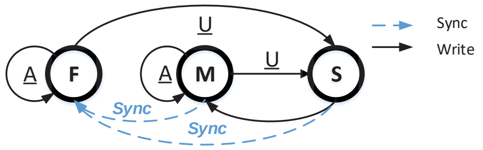


Figure 5: The State Predication Diagram. Each write request can only be one of the three write states, F, M, and S. Letter "$\underline{A}$" means that the current data written in the F and M states are less than the W_f and W_m values, respectively. Otherwise, the write buffer is "$\underline{U}$". The *Sync* operation takes the next write state back to F.

executing slow writes with padded non-user data.

To overcome the second issue, we propose the SSD-HDD Mixed IO scheduler (**MIOS**) that adaptively controls queue lengths of SSDs and HDDs in hybrid storage nodes, and determines where to steer a user write request.

4.1 Write-state Predictor

The next write state could be predicted according to write buffer’s free space and the write state of the current request. In HDD buffered write, each write request state should be one of F, M, and S state. The HDD write buffer state is considered by buffered-write model to be in either A (available) or U (unavailable). The "A" state means that the current Accumulative Data Written (**ADW**) in the F and M states are less than W_f and W_m, respectively. Otherwise, the write buffer is in the "U" state. Figure 5 shows how the next write state is determined based on the current buffer state and write state in a State Predication Diagram, which is described as follows:

- F/A : The current write state is F and the buffer is available. Next write state is most likely to be F.
- F/U : Although the current write state is F, the buffer is unavailable. Next write state is likely to change to S.
- M/A : The current write state is M and the buffer is available. Next write state is most likely to remain M.
- M/U : Although the current write state is M, the buffer is unavailable. Next write state should be S.
- S : The current write state is S. Next write state will be M with a high probability.
- The *Sync* operation will force the next write state and buffer state back to be F/A in all cases.

Based on that, we design a Write-state Predictor described in Algorithm 1. It identifies what the current write state is, F, M or S, by monitoring the IO request size and latency, and calculating the free space in the write buffer. That is, the ADW in the current write state (F or M) is recorded and compared with W_f or W_m, for predicting the next write state.

Next, we assess the prediction accuracy of the write-state predictor. We write 100GB data with an IO size of 128KB to the 10TB WD HDD and invoke *sync()* after each 1GB data written. The results show that the predictor correctly identifies 99.5% of the F state, 98.1% of the M state and 60.3% of the S state. The low prediction accuracy for the S write-state is

Algorithm 1 The algorithm of Write-state Predictor

```
Input: Current write request size: $size$; The last write state: $state$; Current accumulative amount of data written: $ADW$;
The amounts of data written in the $F$ state and $M$ state: $W_F$ and $W_M$
Output: Write-state prediction for the next request ($F$, $M$ or $S$ )
 1: function Predictor()
 2: if state == $F$ then
 3:     if (ADW + size) < $W_f$ then : return $F$
 4:     else: return $S$
 5:     end if
 6: else if state == $M$ then
 7:     if (ADW + size) < $W_m$ then : return $M$
 8:     else: return $S$
 9:     end if
10: else: return $M$
11: end if
```

due to the prediction policy that tends to favor the S write-state to reduce the performance degradation, when an actual S write-state is mis-predicted as a different state.

4.2 Buffer-Controlled Writes

Buffer-Controlled Writes (BCW) is an HDD writing approach that ensures user writes using F or M write state, and avoids allocating Slow writes. The key idea of BCW is to make buffered write controllable. Based on the Write-state predictor, we design BCW, as described in Algorithm 2.

Upon activating BCW, a $sync()$ operation is invoked to force synchronization to empty the buffer actively. BCW dispatches sequential user writes to HDD if it is predicted to be in F or M state, otherwise pads non-user data to HDD, until it reach to the max setting loop (or unlimited) of Buffered Write Sequence. If there are user requests in the queue, BCW writes them serially. After a write is completed, BCW adds its write size to ADW, and updates the write-state accordingly.

During light or idle workload periods with sparse requests, the HDD request queue will be empty from time to time, making the write stream discontinuous. To ensure the stability and certainty of buffered writes in a sequential and continuous pattern, BCW will proactively pad non-user data to write to the disk. The padding data are of two types, PF and PS. The former is used to fill the F and M states with 4KB non-user data; the latter is to fill the S state with larger block size, e.g., 64KB of non-user data. A small PF can minimize the waiting time of user requests. A large PS helps trigger Slow write quickly. Note that even for each padded write, BCW still executes the write-state predictor algorithm.

More specifically, BCW continuously calculates ADW in the current state (F or M). When ADW is close to W_f or W_m, it means that the HDD buffered write is at the tail of the Fast or Mid stage. The S write state may occur after several writes. At this point, BCW notifies the scheduler and proactively triggers the Slow write with PS.

To avoid long latency for user writes, at this period, all incoming user requests have to be steered to other storage devices, such as SSDs. When an S write completed, the next

Algorithm 2 The algorithm of Buffer-Controlled Write

```
Input: The max loop of Buffered Write Sequence: $Loop_{max}$
Request $R_i$ size: $size_i$; Current written amount: $ADW$;
The state of last write: $state$;
Active padded writes and their size: $PS$,$PF$ and $size_{PS}$,$size_{PF}$
 1: $sync()$
 2: while $loop < Loop_{max}$ do
 3:     if request $R_i$ in the HDD write queue then
 4:         write $R_i$ to HDD, update $ADW$ and $state$
 5:     else
 6:         if $Predictor() == S$ then
 7:             $flag_{HDD}$ = False    // Stop receiving
 8:             while $state == S$ do
 9:                 write $PS$ to HDD, update $ADW$ and $state$
10:             end while
11:             $flag_{HDD}$ = True    // Start receiving
12:             reset $ADW$; $loop$++
13:         end if
14:         if $Predictor() == M$ then
15:             write $PF$ to HDD; update $ADW$ and $state$
16:         end if
17:     end if
18: end while
```

write will be M according to the write-state predictor. Then BCW resets the ADW and accepts user requests again.

We also find it unnecessary to proactively fill padded writes in the Fast state before ADW exceeds W_f. When ADW does not reach W_f, the physical disk operation is not yet triggered and the buffer can absorb user requests for this period. When ADW exceeds W_f in a short time of period, it means that the buffer will begin to flush the data to the disk and the next write state will be changed to S. On the other hand, when ADW is less than W_f for a long time of period, the disk can flush the buffered data automatically so that the next write state may be F. However, it does not affect performance. We apply this observation to the scheduler design in the next section.

In most cases, the sequential and continuous write pattern induced by BCW is reasonably stable. However, this pattern can be broken, e.g., HDD reads. Besides, slow writes may be triggered in advance by user requests or PF writes. To regain the control of buffered writes, BCW continuously executes PS until a S write state is detected. As a result, the write-state predictor will be recalibrated. The cost of this strategy is that BCW wastes IO and storage of HDD to perform PS writes. In addition, we also can issue $sync()$ to reset the buffered write state in BCW. Howerver, a $sync()$ can take several hundred milliseconds, during which the HDD cannot accept any writes. Fortunately, in the experiment, we found that the BCW interrupted cases are rare.

BCW stores incoming data to HDD in log-append manner. This differs with traditional logging mode in existing file systems like ext4 [33]. The latter allocates writes to the tail logical address of the log, ensuring address continuity. However, it doesn't ensure IO continuity and does not determine the next write state. In contrast, BCW maintains both address and IO continuity, make the buffer writing be controllable.

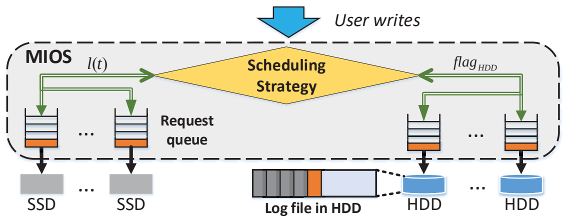


Figure 6: Architecture of the Mixed IO scheduler. It monitors all request queues of SSDs and HDDs. The user writes meeting the conditions are redirected to appropriate HDDs.

Algorithm 3 The algorithm of Mixed IO Scheduler

```
Input: SSD queue length at time $t$: $l(t)$;
Queue length threshold: $L$; HDD available flag: $flag_{HDD}$;
Schedule Strategy: $MIOS\_D$ or $MIOS\_E$
1: if ($flag_{HDD}$ == True) then
2:     if $l(t) > L$ && $Predictor() == F$ or $M$ then
3:         Send to HDD queue
4:     else if $MIOS\_E$ && $Predictor() == F$ then
5:         Send to HDD queue
6:     else: Send to SSD queue
7:     end if
8: else: Send to SSD queue
9: end if
```

4.3 Mixed IO scheduler

BCW provides a proactive and controllable buffer writing approach. In this section, we further propose a *Mixed IO scheduler* (**MIOS**) for SSD-HDD hybrid storage to leverage BCW effectively. The scheduler decides whether or not to steer user writes to a HDD request queue according to the results of the Write-state Predictor and current queue status.

Architecture The architecture of MIOS is shown in Figure 6. MIOS monitors all request queues of SSDs and HDDs at runtime, judiciously triggers the BCW process, and determines whether a user write should be directed to a selected HDD or SSD. MIOS creates a device file in each HDD in the configuration process. The device file stores BCW writes in an append-only manner. Before MIOS scheduling, a Profiling is performed to determine the key parameters (W_f, W_m, etc.) for the write-state predictor.

Scheduling Strategy In algorithm 3, the SSD request queue length $l(t)$ at time t is a key parameter in MIOS. When $l(t)$ is larger than a predefined threshold L, the scheduler steers user writes to an HDD with the prediction of it being F or M write state. The threshold L is pre-determined according to the actual performance measurements on SSD. Specifically, we measure the write latency under different SSD queue lengths. If the request with queue length l has latency larger than the IO delay of HDD in the M state, we simply set the threshold L to the minimum l. The rationale is that when the SSD queue length is larger than L, the SSD writes' latencies will be at the same level as their latencies on an HDD in the F or M write state with BCW. L can be determined and adjusted experimentally according to workload behaviors and storage device configurations at runtime. This strategy mitigates, though not avoids, the long-tail latency upon workload bursts or heavy Garbage Collections on SSD [37,48]. In these cases, the SSD request queue length can be 8-10 times longer than its average. Therefore, redirected HDD writes not only relieve SSD pressure imposed by bursty requests and heavy GCs, curbing the long-tail latency, but also lower the average latency.

Additionally, when the queue length of SSD is less than L, triggering BCW is optional. Enabling or disabling BCW in this case is denoted as **MIOS_E** or **MIOS_D**, respectively. In other words, *MIOS_E* strategy allows redirection with BCW when the queue length of SSD is lower than L. *MIOS_D* strategy, by contrast, disables redirection when the SSD queue length is lower than L. Note that the write latency of an HDD in the M write state is still much higher than that of an SSD. The request latency after redirection may be increased. Therefore, when the $l(t)$ is lower than L, we only redirect user requests to leverage the F write state of HDD in *MIOS_E*. We will experimentally analyze the positive and negative effects of *MIOS_D* and *MIOS_E* in Section 5.

Generally, a typical hybrid storage node contains multiple SSDs and HDDs. We divide all disks into independent SSD/HDD pairs, each of which contains an SSD and one or more HDDs. Each SSD/HDD pair is managed by an independent MIOS scheduler instance.

Finally, MIOS requires the complete control over HDDs. It means that the HDDs in BCW cannot be interfered by other IO operations. When an HDD is executing BCW and a read request arrives, **MIOS** immediately suspends BCW and serves this read. It will try to redirect all writes to other idle disks at this time. For read-dominated workloads, BCW can be disabled to avoid interfering with reads.

4.3.1 Implementation

MIOS can be implemented in either file-level or volume-level to jointly manage SSDs and HDDs in a hybrid storage. In this work, MIOS provides a simple yet flexible file-level request scheduling scheme atop of existing file systems to leverage their mature file-to-storage mapping mechanism. A user request is identified with the corresponding filename and file internal offset. To reduce overhead of the underlying file system, MIOS employs direct IO mode to access the log by calling Linux kernel functions such as *open*, *close*, *read*, and *write*. Data of each write request is stored as a chunk in the log. We design a metadata structure that records and tracks chunks in the log. We choose a hash table and use the file ID field of a request as the hash key.

When an HDD is idle, all user data stored in the log will be written to their own original files, after which the log will be recycled, called as HDD Garbage Collection. HDD GC should be triggered when the log size exceeds a predefined threshold (e.g., 70% capacity of HDD). HDD GC first sequentially and continuously reads user data chunks that are interspersed with

Table 3: The amount of redirected writes data and requests with the *MIOS_D* and the *MIOS_E* strategies.

Workload Type	A	B	C	D
Writing Method	Baseline / MIOS_D / MIOS_E			
SSD Writes (GB)	14.7 / 13.9 / 1.2	61.2 / 57.1 / 48.1	7.2 / 6.1 / 2.1	7.5 / 6.3 / 2.1
HDD Writes (GB)	- / 4.1 / 61.6	- / 18.4 / 56	- / 4.5 / 22.3	- / 4.4 / 25.6
SSD Requests (millions)	0.43 / 0.36 / 0.04	4.4 / 3.7 / 1.3	4.8 / 3.7 / 1.6	4.7 / 3.8 / 1.3

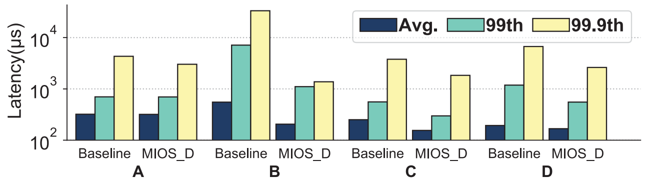


Figure 7: The average, 99th and 99.9th-percentile latency under four Pangu production workloads, comparing *Baseline* with *MIOS_D* (a logscale is used for the y-axis).

padded data in the log to reduce seeks. And then it extracts and merges the user data to update their correspond files. These file updates can be performed in batch [53].

5 Evaluation

5.1 Experiment Setup

We run experiments for performance evaluation on a server with two Intel Xeon E5-2696 v4 processors (2.20 GHz, 22 CPUs) and 128 GB of DDR3 DRAM. To understand the impact of different storage configurations on the performance, we choose two types of SSDs, a 256GB Intel 660p SSD [11] and a 256GB Samsung 960EVO SSD [16]. Their peak write throughputs are 0.6 GB/s and 1.5GB/s, respectively. Three types of HDDs are WD 10TB, WD 4TB, and Seagate 4TB, as described in Section 3.1.

The 10TB WD HDD has a W_f of 16MB and W_m of 8MB. Using the process to pre-determine the queue length threshold L explained in Section 4, we set L to 1 for workload of node A, 3 for node B, 2 for node C and D, where the workloads of nodes A, B, C and D are described in Table 1 of Section 2. As discussed earlier, MIOS has two schemes, **MIOS_D** and **MIOS_E**. When the SSD queue length is less than L, the former conservatively disables request redirection; the latter allows request redirection but only redirects user write requests to the F write state. The **Baseline** for the evaluation is writing all user data into the SSDs. In addition, a complete BCW sequence consists a series of 1 Fast stage and 10 Mid/Slow stage-pairs (Figure 4).

5.2 MIOS under Production Workloads

We first evaluate the effectiveness of *MIOS_D* under four Pangu production workloads on the WD 10TB HDD.

Write Performance Figure 7 shows that the average and tail latency (99th and 99.9th) of all four workloads are significantly reduced by *MIOS_D*. Among four workloads, B gains the most benefit. Its average, 99th and 99.9th-precentile

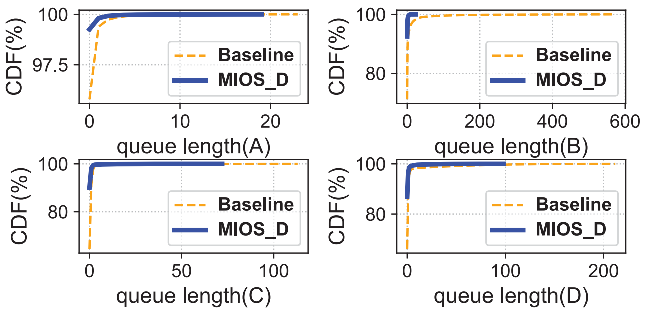


Figure 8: The CDF of SSD queue length.

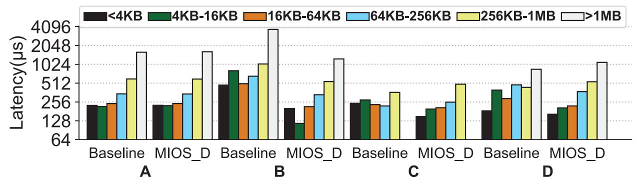


Figure 9: The average request latency in six request-size groups that are classified by IO size with *MIOS_D*.

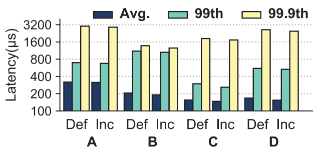
latencies are reduced by 65%, 85%, and 95% respectively. On the contrary, these three latencies in A are only reduced by about 2%, 3.5% and 30%, respectively, which is far less than the other workloads. The reason is that the redirection in *MIOS_D* is only triggered when the queue length is high, but *A* has the least intensity and thus the least queue blocking, which renders MIOS much less useful.

To better understand the root causes for the above experimental results, the cumulative distribution functions (CDFs) of SSD queue lengths for four workloads are shown in Figure 8. *MIOS_D* significantly shorten queue lengths compared to *Baseline*. B and A have the maximum (95%) and minimum (15%) reduction in their queue lengths. Therefore, *MIOS_D* reduces the overall queueing delay significantly.

Request size To deeply understand impact of write size in *MIOS_D* and BCW, we break down all redirected requests into six groups with different ranges of IO sizes, and measure the average latency in each group.

Figure 9 shows that, in all four workloads, *MIOS_D* reduces the average write latency of size below 64KB. The B workload benefits the most. The average latencies of three groups of small-sized requests (<4KB; 4KB-16KB; 16KB-64KB) are reduced by 61%, 85%, and 59%, respectively. The other three workloads also reduce their latencies differently. In *Baseline*, small and intensive requests result in queue blocking more frequently (Figure 2) than in *MIOS_D*. Therefore, *MIOS_D* is the most effective in reducing latency in such cases.

However, in groups of requests larger than 256KB, the average latency is increased in all workloads except B. The latency is increased by up to 31.7% in the >1MB group, and 12.1% in the 256KB-1MB group for the D workload. The average latency of the 256KB-1MB group in C is increased by 20.1%. The reason is twofold. First, large SSD writes under light load have better performance than HDDs because


(a) The average and 99^{th} tail latency with different L values

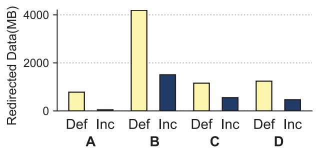
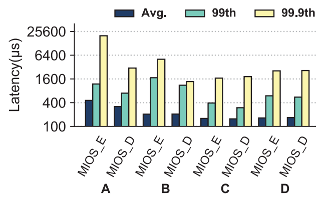

(b) The redirected written data with different L value

Figure 10: *MIOS_D* with different queue length threshold *L*.


(a) The average and tail latency with *MIOS_D* and *MIOS_E*

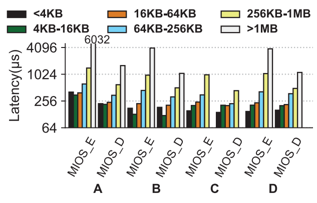

(b) The redirected written data for different request-size group

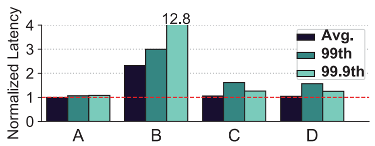
Figure 11: *MIOS_D* vs *MIOS_E*.

SSDs have high internal-parallelism that favors large requests. Second, large writes are relatively sparse and not easy to be completely blocked. For example, the average latency of the >256KB request-size groups in *Baseline* is very close to the raw SSD write performance.

Queue Length Threshold *L* To evaluate the effect of *L* selection, we compare the pre-defined *L* value (**Def**), determined by the process described Section 4.2, with $L+1$ (**Inc**). Note that the process for pre-defining the queue length threshold is designed to tradeoff between reducing the write latency and reducing the write traffic to SSD.

Figure 10(a) shows that, *Inc* slightly reduces average, 99^{th} and 99.9^{th}-percentile latency compared to *Def*. Among the four workloads, the maximum reduction in average latency is less than 10%. This is because the higher queue length is, the longer waiting delay a request experiences. Therefore, *Inc* can acquire more latency gains by redirection than *Def*. However, the choice of *L* value can greatly affect the amount of redirected data. In Figure 10(b), the number of redirected requests is much smaller in *Inc* than in *Def*. The amount of redirected data for workloads A∼D are decreased by 94%, 64%, 52% and 62%, respectively. These results are consistent with the implications of Figure 8 that longer queue length in SSD triggers much fewer SSD overuse alerts, significantly reducing chances for request redirecting to HDD.

MIOS_D vs MIOS_E We compare *MIOS_D* with *MIOS_E* in terms of the amount of data written to SSD and HDD, and the number of redirected write requests, as shown in Table 3. Workload A has the highest percentage of data and requests redirected with *MIOS_E*, reducing the SSD written data by up to 93.3% compared with *Baseline*, which is significantly higher than *MIOS_D*. Since workload A has lower IO intensity, *MIOS_E* has more chances to redirect even when the queue length is low. Note that we also counted the padded data in BCW as the amount of data written in HDD. In such a case the total amount of data written can vary a great deal. Workload B has the lowest percentage of redirection with *MIOS_E*, which reduces SSD written data by 30%. Nevertheless, the absolute amount of redirected data is very large because the SSD written data in *Baseline* is larger than any of the other three workloads. Compared with *MIOS_D*, *MIOS_E* can greatly decrease the amount of data written to SSD. Therefore, it is beneficial to alleviate SSD wear-out.


(a) Average and tail latency

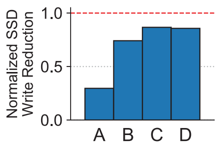

(b) SSD write reduction

Figure 12: Latency and SSD written data reduction with only *F* write-state by actively issuing sync() (Normalized to *MIOS_D*).

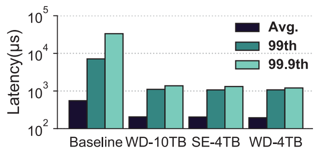
Table 4: Amount of data written to and number of requests processed in SSD with different HDDs under workload B.

	Baseline	WD-10TB	WD-4TB	SE-4TB
SSD written data (GB)	61.2	4.1	4.2	4.4
SSD write requests (thousands)	4453	720	724	769

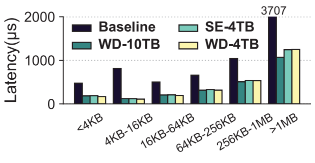
However, the negative effect of *MIOS_E* is the increase of average and tail latency. In Figure 11(a), *MIOS_E* leads to generally higher average latency than *MIOS_D* by up to 40% under workload A. Although for the other three workloads, the average latency remains basically unchanged. This is because for workload A much more writes (i.e., >90%) are redirected by *MIOS_E* than by *MIOS_D*, and in HDD requests experience longer latency than in SSD. Moreover, the 99.9^{th}-percentile latency of *MIOS_E* is increased by 70% in A, 55% in B, 31% in C, and 8% in D compared to *MIOS_D*. The results can be explained by Figure 11(b). *MIOS_E* increases the average latency for nearly all the IO size groups, especially for the groups with requests of size larger than 256KB.

Moreover, we only use the *F* write state by proactively issuing *sync*() when the ADW reaches W_f. In Figure 12, we measure the average, 99^{th}, 99.9^{th}-percentile latency and the SSD written data reduction with this strategy. We take the MIOS_D as the baseline and present the performance normalized to MIOS_D. The 99.9^{th}-percentile latency is increased by 12.8x over MIOS_D in the B node. The 99^{th}-percentile latency in the B, C and D nodes also be increased by 3x, 1.65x and 1.56x, respectively. This means that this strategy is less efficiency for reducing tail latency when the workload becomes heavier. This is because it redirects less SSD write data than MIOS_D when SSD suffers queue blockage. As mentioned in Section 4.2, *sync*() is a high cost operation (e.g., several hundreds of milliseconds) to flush the HDD buffer and cannot serve any requests during the operation.

Experiment with other HDDs We use the 4TB WD, 4TB Seagate and the 10TB WD HDD to replay workload B, comparing *MIOS_D* (with the default *L* value) with the *Baseline* in terms of the amount of data written to and the number of write requests processed in SSD. Workload B is chosen for

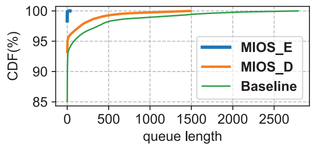


(a) Average, 99^{th} and 99.9^{th}-percentile latency

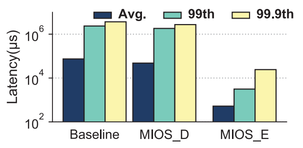


(b) Average latency of requests with different request-size group

Figure 13: *MIOS_D* performance with three different types of HDDs under workload B.



(a) Queue length CDF



(b) Average and tail latency

Figure 14: Queue length CDF and latency under Pangu workload A, with the 660p SSD for three scheduling strategies.

this experiment, since it has the most SSD written data and the most severe SSD queue blockage, clearly reflecting the effect of IO scheduling.

Figure 13 shows that different types of HDDs do not have a significant impact on the effect of *MIOS_D*. First, the average and tail latencies for all the three HDDs are virtually identical, with a maximum difference of less than 3%. In addition, of the six request-size groups, only the >1MB group exhibits a large difference among the different HDDs. The average latency of 10TB HDD is 14% lower than that of the other two 4TB HDDs. This is because the native write performance gap between the HDDs. It can be found from Table 4 that different types of HDDs do not notably affect the amount of data redirected, with little difference of less than 5%.

Experiment with lower-performing 660p SSD Next, to further explore the effect of MIOS with different SSDs, we deploy the lower-performance 660p SSD. We replay the same workload A that has the lowest pressure on SSD, and employ the *MIOS_D* and *MIOS_E* strategies, respectively.

From the latency CDF Figure 14(a), when using SSDs with the low performance SSD, more than 7% of the requests are severely affected by long queuing delay and the maximum queue length reaches up to 2700. It surpass the experiment result with 960EVO SSD (e.g., 23 shown in Figure 8(B)). This is because when the IO intensity exceeds the ability of 660p SSD to accommodate, the SSD queue length builds up quickly. As a result in Figure 14(b), the average and tail latencies in *Baseline* rise sharply compared with 960EVO SSD shown in previous Figure 7. The average latency in *Baseline* is 90ms and the 99^{th}-percentile latency exceeds 5 second.

With such a high workload pressure on a lower-performance SSD, MIOS can help reduce some of the pressure on SSD by redirecting some of the queued requests to HDDs. As seen from Figure 14, *MIOS_D* decreases the queue blockage with a maximum 45% queue length reduction. At

Table 5: The HDD utilization with *MIOS_E* and *MIOS_D*.

Node Type	Duration (s)	Baseline	Net Util. MIOS_D	Net Util. MIOS_E	Gross Util. MIOS_E
A	2700	7.6%	7.9%	11.9%	27.9%
B	1800	9.8%	18.2%	26.8%	56.9%
C	1800	4.1%	10.7%	16.2%	35.8%
D	1560	4.8%	12.3%	17.3%	39.5%

same time, the average latency in *MIOS_E* returns to μs-level (e.g., 521μs), and the 99^{th} and 99.9^{th}-percentile latencies are reduced to an acceptable range of 2.4ms and 87ms, respectively. Because *MIOS_E* redirects much more SSD requests with low queue length, it prevents queue blockage in SSD, particularly for a lower-performance one. By comparing this experiment on a lower-performance SSD with an earlier one on a high-performance SSD, we believe that when the performance of SSD in hybrid storage cannot support the intensity of a write-dominated workload, MIOS and BCW can provide an effective way to improve the overall IO capacity by offloading much of the workload pressure on SSD to HDD.

In addition, we compare BCW to a system that simply adds an extra SSD. We equally distribute the workloads to two SSDs. The system can achieve the same or even better latency than *MIOS_E*, but at a significantly increased hardware cost.

5.3 BCW

We further analyze and evaluate the wasted storage-space of BCW, due to the padded data to help keep continuous HDD written pattern and skip the *S* write state.

Amount of padded data We first analyze the amount of padded data written to HDD. In Table 3, we measure the data amount with *MIOS_D* and *MIOS_E* when a BCW sequence contains one Fast state and 10 Mid/Slow stage-pairs. The stats in the table clearly indicates that *MIOS_E* generates substantially more HDD write data than *MIOS_D*. When more requests are redirected, the amount of padded data increases proportionally. For example, the padded data with *MIOS_E* is 15x that with *MIOS_D* in workload A, 3x in workload B, 4x in workloads C and D. Frequently triggering BCW increases the occurrences and thus amount of padded data. Furthermore, when the amount of redirected data increases, the Fast stage without padded data will be used up faster and more Mid/Slow stage-pairs with padded data will be executed.

HDD utilization The original Pangu traces exhibit low HDD utilization, which is defined by the percentage of time an HDD is actually working on processing IO requests. More specifically, Table 1 and 5 shows that HDDs are generally keeps very low utilization (e.g., <10%) in all four workloads.

Using MIOS, the HDD utilization has been increased with different degrees. The gross utilization is defined to be the percentage of the total execution time when the HDD is working on IO requests (including *sync*() operation), which is the real usage of the disk. The highest gross utilization is 56.9% under workload B. This means that the disk still has enough free time for HDD garbage collection. To analyze the amount of

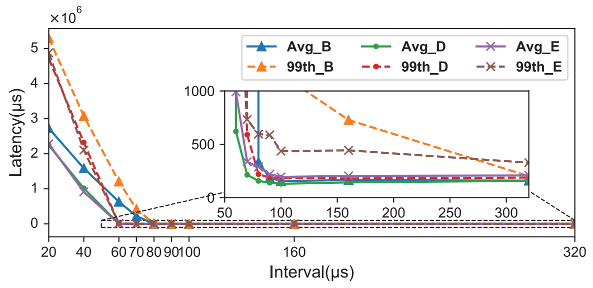


Figure 15: FIO benchmark to experiment with three strategies. The IO transmission interval is set to 20-320us.

time HDD is effectively working for user requests, we define *net utilization* as the percentage of the total execution time that the HDD spends exclusively serving user requests, excluding the time HDD spends on padding data in BCW. Thus, the net utilization is positively correlated to the amount of redirected data. The net utilization of HDD in *MIOS_E* is higher than that in *MIOS_D*. Under workload B and *MIOS_E*, HDD has the highest net utilization improvement over *Baseline*, by 2.7x, while the same is enhanced to 1.8x under *MIOS_D*.

5.4 Write Intensity

The effectiveness of BCW heavily depends on the write intensity. To better understand this relationship, we test the average and tail latency of three scheduling strategies as a function of write intensity (in terms of IO transmission interval), *Baseline* (**_B**), *MIOS_D* (**_D**) and *MIOS_E* (**_E**). We initialize the IO size to 32KB, and continuously issue write requests using FIO [4]. Since FIO cannot adjust IO intensity, we set the generated IOs with a fixed transmission interval from 20 to 320μs. We use 960EVO SSD and 10TB HDD, and set the L value to 1.

Figure 15 shows that, when the interval is between 20-60μs, the requests written to SSD are severely blocked and the 99th tail latency reaches as high as 5.2 second. In this case, both *MIOS_D* and *MIOS_E* can significantly reduce the request latency. And *MIOS_E* is slightly better than *MIOS_D* because the former handles burst writes better. When the interval is 60-80μs, *Baseline* still exhibits very high latency in SSD. However, the latency has returned to an acceptable μs-level after scheduling by MIOS. When the interval exceeds 100μs, the average and 99th-percentile latencies are stable, because there is very little SSD queue blockage with this level of request intensity. In this case, *MIOS_D* and *Baseline* have the lowest average latency and remain the same as the interval grows. However, the average and tail latency of *MIOS_E* is higher than others. This is because even if there is no queue in SSD, *MIOS_E* will still redirect requests, and the performance gap between SSD and HDD can lead to high latency.

6 Related Works

IO scheduler The IO scheduling on HDD had been adequately studied as CFQ, Anticipatory, Deadline [7], and NCQ [54]. With wide adoption of SSDs, more recent researches address flash IO characteristics as read/write performance asymmetry and internal parallelism. FIOS [39] employs a fair IO timeslice management to attains fairness and high efficiency of SSD. HIOS [23] gives GC-aware and QoS-aware scheduler in host. PIQ [19] and ParDispatcher [46] minimize access conflicts between IO requests. A large body of research further offer finer scheduling inside of SSD to reduce interference between IO flows [42], write amplification [27], and GC overhead [17,21,24]. SWAN [26] partitions SSDs into multiple zones to separately serve write requests and perform GC. These works focus on homogeneous-device block-level scheduling. In contrast, MIOS schedules writes upon SSD-HDD hybrid storage.

Hybrid storage For SSD-HDD hybrid storage, most works use SSDs as a read cache or/and write buffer [2,25,41], and HDDs as the secondary or backup storage [29], due to the large performance gap between SSD and HDD. Prior works also employ HDDs as a write cache for SSDs to reduce the amount of data written to the latter [41,52]. Besides, SSD-HDD mixed RAID [32] also has been studied to complement their disadvantages with advantages. Ziggurat [55] as a tiered file system across NVMM and disks steers larger asynchronous writes into disks. SWR [31] merely redirects synchronous large writes to HDDs at highly queueing. BCW further exploits HDD buffer to redirect synchronous small writes while avoiding performance degradation.

7 Conclusion

Some hybrid storage servers serve write-dominate workloads, which leads to SSD overuse and long-tail latency while HDDs are underutilized. However, our extensive experimental study reveals that HDDs are capable of μs-level write IO latency with appropriate buffered writes. This motivated us to use HDDs to offload write requests from overused SSDs by request redirection. To this end, we present a Buffer-Controlled Write approach to proactively control buffered writes, by selecting fast writes for user requests and padding non-user data for slow writes. Then, we proposed a mixed IO scheduler to automatically steer incoming data to SSDs or HDDs based on runtime monitoring of request queues. Our extensive evaluation of MIOS and BCW, driven by real-world production workloads and benchmarks, demonstrated their efficacy.

Acknowledgments

We would like to thank our shepherd, Jian Huang, and the anonymous reviewers for their valuable feedback and suggestion. This work is supported in part by NSFC No.61821003, NSFC No.61872156, National key research and development program of China (No.2018YFA0701804), the US NSF under Grant No.CCF-1704504 and No.CCF-1629625, and Alibaba Group through Alibaba Innovative Research (AIR) Program.

References

[1] David G Andersen and Steven Swanson. Rethinking flash in the data center. *IEEE micro*, 30(4):52–54, 2010.

[2] Manos Athanassoulis, Shimin Chen, Anastasia Ailamaki, Phillip B. Gibbons, and Radu Stoica. Masm: efficient online updates in data warehouses. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 865–876, 2011.

[3] Guillaume Aupy, Olivier Beaumont, and Lionel Eyraud-Dubois. What size should your buffers to disks be? In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 660–669. IEEE, 2018.

[4] AXBOE. Fio: Flexible i/o tester. https://github.com/axboe/fio.

[5] Oana Balmau, Florin Dinu, Willy Zwaenepoel, Karan Gupta, Ravishankar Chandhiramoorthi, and Diego Didona. SILK: Preventing latency spikes in log-structured merge key-value stores. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 753–766, Renton, WA, July 2019. USENIX Association.

[6] Simona Boboila and Peter Desnoyers. Write endurance in flash drives: Measurements and analysis. In *FAST*, pages 115–128, 2010.

[7] Daniel P Bovet and Marco Cesati. *Understanding the Linux Kernel: from I/O ports to process management*. " O'Reilly Media, Inc.", 2005.

[8] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157. ACM, 2011.

[9] Alibaba Clouder. Pangu – the high performance distributed file system by alibaba cloud. 2018. https://www.alibabacloud.com/blog/pangu_the_high_performance_distributed_file_system_by_alibaba_cloud_594059.

[10] Intel Corporation. Enterprise-class versus desktop-class hard drives. pages 6–7, 2016. https://www.intel.com/content/dam/support/us/en/documents/server-products/Enterprise_vs_Desktop_HDDs_2.0.pdf.

[11] Intel Corporation. Product brief of intel 660p series. pages 2–2, 2019. https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/660p-series-brief.pdf.

[12] Western Digital Corporation. Product brief: Wd gold enterprise class sata hdd. pages 2–3, 2019. https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/product/internal-drives/wd-gold/product-brief-wd-gold-2579-810192.pdf.

[13] Western Digital Corporation. Wd red nas hard drives data sheet. pages 2–3, 2019. http://products.wdc.com/library/SpecSheet/ENG/2879-800002.pdf.

[14] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS operating systems review*, volume 41, pages 205–220. ACM, 2007.

[15] Diego Didona and Willy Zwaenepoel. Size-aware sharding for improving tail latencies in in-memory key-value stores. In *NSDI*, pages 79–94, 2019.

[16] Samsung Electronics. Samsung ssd 960 evo m.2 data sheet. pages 2–4, 2017. https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/660p-series-brief.pdf.

[17] Nima Elyasi, Mohammad Arjomand, Anand Sivasubramaniam, Mahmut T Kandemir, Chita R Das, and Myoungsoo Jung. Exploiting intra-request slack to improve ssd performance. *ACM SIGARCH Computer Architecture News*, 45(1):375–388, 2017.

[18] FUJITSU. Mbc2073rc mbc2036rc hard disk drives product manual. pages 60–62, 2007. https://www.fujitsu.com/downloads/COMP/fel/support/disk/manuals/c141-e266-01en.pdf.

[19] Congming Gao, Liang Shi, Mengying Zhao, Chun Jason Xue, Kaijie Wu, and Edwin H-M Sha. Exploiting parallelism in i/o scheduling for access conflict minimization in flash-based solid state drives. In *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–11. IEEE, 2014.

[20] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. 2003.

[21] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings. volume 44. ACM, 2009.

[22] Congfeng Jiang, Guangjie Han, Jiangbin Lin, Gangyong Jia, Weisong Shi, and Jian Wan. Characteristics of co-allocated online services and batch jobs in internet data centers: A case study from alibaba cloud. *IEEE Access*, 7:22495–22508, 2019.

[23] Myoungsoo Jung, Wonil Choi, Shekhar Srikantaiah, Joonhyuk Yoo, and Mahmut T Kandemir. Hios: a host interface i/o scheduler for solid state disks. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 289–300. IEEE Press, 2014.

[24] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. The multi-streamed solid-state drive. In *6th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*, 2014.

[25] Taeho Kgil and Trevor Mudge. Flashcache: a nand flash memory file cache for low power web servers. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 103–112. ACM, 2006.

[26] Jaeho Kim, Kwanghyun Lim, Youngdon Jung, Sungjin Lee, Changwoo Min, and Sam H Noh. Alleviating garbage collection interference through spatial separation in all flash arrays. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 799–812, 2019.

[27] Jaeho Kim, Yongseok Oh, Eunsam Kim, Jongmoo Choi, Donghee Lee, and Sam H Noh. Disk schedulers for solid state drivers. In *Proceedings of the seventh ACM international conference on Embedded software*, pages 295–304. ACM, 2009.

[28] Anthony Kougkas, Hariharan Devarajan, and Xian-He Sun. Hermes: a heterogeneous-aware multi-tiered distributed i/o buffering system. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pages 219–230. ACM, 2018.

[29] Huiba Li, Yiming Zhang, Dongsheng Li, Zhiming Zhang, Shengyun Liu, Peng Huang, Zheng Qin, Kai Chen, and Yongqiang Xiong. Ursa: Hybrid block storage for cloud-scale virtual disks. In *Proceedings of the Fourteenth EuroSys Conference 2019*, page 15. ACM, 2019.

[30] Qixiao Liu and Zhibin Yu. The elasticity and plasticity in semi-containerized co-locating cloud workload: A view from alibaba trace. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 347–360. ACM, 2018.

[31] Shuyang Liu, Shucheng Wang, Qiang Cao, Ziyi Lu, Hong Jiang, Jie Yao, Yuanyuan Dong, and Puyuan Yang. Analysis of and optimization for write-dominated hybrid storage nodes in cloud. In *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2019, Santa Cruz, CA, USA, November 20-23, 2019*, pages 403–415, 2019.

[32] Bo Mao, Hong Jiang, Suzhen Wu, Lei Tian, Dan Feng, Jianxi Chen, and Lingfang Zeng. Hpda: A hybrid parity-based disk array for enhanced performance and reliability. *ACM Transactions on Storage (TOS)*, 8(1):4, 2012.

[33] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. The new ext4 filesystem: current status and future plans. In *Proceedings of the Linux symposium*, volume 2, pages 21–33, 2007.

[34] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. Sfs: random write considered harmful in solid state drives. In *FAST*, volume 12, pages 1–16, 2012.

[35] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, et al. f4: Facebook's warm {BLOB} storage system. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 383–398, 2014.

[36] Muthukumar Murugan and David HC Du. Rejuvenator: A static wear leveling algorithm for nand flash memory with minimized overhead. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12. IEEE, 2011.

[37] J. Ou, J. Shu, Y. Lu, L. Yi, and W. Wang. Edm: An endurance-aware data migration scheme for load balancing in ssd storage clusters. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 787–796, May 2014.

[38] Mayur R Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. Amazon s3 for science grids: a viable solution? In *Proceedings of the 2008 international workshop on Data-aware distributed computing*, pages 55–64. ACM, 2008.

[39] Stan Park and Kai Shen. Fios: a fair, efficient flash i/o scheduler. In *FAST*, volume 12, pages 13–13, 2012.

[40] Raghu Ramakrishnan, Baskar Sridharan, John R Douceur, Pavan Kasturi, Balaji Krishnamachari-Sampath, Karthick Krishnamoorthy, Peng Li, Mitica Manu, Spiro Michaylov, Rogério Ramos, et al. Azure data lake store: a hyperscale distributed file service for

big data analytics. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 51–63. ACM, 2017.

[41] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. Extending ssd lifetimes with disk-based write caches. In *FAST*, volume 10, pages 101–114, 2010.

[42] Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan Gómez-Luna, and Onur Mutlu. Flin: Enabling fairness and enhancing performance in modern nvme solid state drives. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 397–410. IEEE, 2018.

[43] Seagate Technology. Enhanced caching advantage—turboboost and advanced write caching. pages 2–3, 2016. https://www.seagate.com/files/www-content/product-content/enterprise-performance-savvio-fam/enterprise-performance-15k-hdd/_cross-product/_shared/doc/enchanced-cache-advantage-tp691.1-1610us.pdf.

[44] Seagate Technology. Barracuda pro compute sata hdd data sheet. pages 2–3, 2018. https://www.seagate.com/www-content/datasheets/pdfs/barracuda-pro-14-tb-DS1901-9-1810US-en_US.pdf.

[45] Seagate Technology. Barracuda compute sata product manual. pages 7–8, 2019. https://www.seagate.com/www-content/product-content/desktop-hdd-fam/en-us/docs/100799391e.pdf.

[46] Hua Wang, Ping Huang, Shuang He, Ke Zhou, Chunhua Li, and Xubin He. A novel i/o scheduler for ssd with improved performance and lifetime. In *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–5. IEEE, 2013.

[47] Hui Wang and Peter Varman. Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation. In *Proceedings of the 12th {USENIX} Conference on File and Storage Technologies ({FAST} 14)*, pages 229–242, 2014.

[48] Yeong-Jae Woo and Jin-Soo Kim. Diversifying wear index for mlc nand flash memory to extend the lifetime of ssds. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, page 6. IEEE Press, 2013.

[49] Erci Xu, Mai Zheng, Feng Qin, Yikang Xu, and Jiesheng Wu. Lessons and actions: What we learned from 10k ssd-related storage system failures. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 961–976, 2019.

[50] Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A Chien, and Haryadi S Gunawi. Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in nand ssds. *ACM Transactions on Storage (TOS)*, 13(3):22, 2017.

[51] Pan Yang, Ni Xue, Yuqi Zhang, Yangxu Zhou, Li Sun, Wenwen Chen, Zhonggang Chen, Wei Xia, Junke Li, and Kihyoun Kwon. Reducing garbage collection overhead in {SSD} based on workload prediction. In *11th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*, 2019.

[52] Puyuan Yang, Peiquan Jin, Shouhong Wan, and Lihua Yue. Hb-storage: Optimizing ssds with a HDD write buffer. In *Web-Age Information Management - WAIM 2013 International Workshops: HardBD, MDSP, BigEM, TMSN, LQPM, BDMS, Beidaihe, China, June 14-16, 2013. Proceedings*, pages 28–39, 2013.

[53] Yang Yang, Qiang Cao, Hong Jiang, Li Yang, Jie Yao, Yuanyuan Dong, and Puyuan Yang. Bfo: Batch-file operations on massive files for consistent performance improvement. In *35th International Conference on Massive Storage Systems and Technology (MSST'19)*, 2019.

[54] Young Jin Yu, Dong In Shin, Hyeonsang Eom, and Heon Young Yeom. Ncq vs. i/o scheduler: Preventing unexpected misbehaviors. *ACM Transactions on Storage (TOS)*, 6(1):2, 2010.

[55] Shengan Zheng, Morteza Hoseinzadeh, and Steven Swanson. Ziggurat: a tiered file system for non-volatile main memories and disks. In *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*, pages 207–219, 2019.